Implementação de um Algoritmo Paralelo para Resolução de Labirintos

Arthur Menezes, Emanoel Aurélio Vianna Fabiano, Gabriell Alves de Araujo

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Partenon, RS, 90619-900 – Porto Alegre – RS – Brazil

{arthur.aguiar, emanoel.fabiano, gabriell.araujo}@acad.pucrs.br

1. Introdução

Dentro do escopo da disciplina de Sistemas Embarcados para o curso de Ciência da Computação, o presente artigo apresenta o desenvolvimento da implementação de um algoritmo paralelo para resolução de labirintos, explorando o modelo de MPSoC do projeto *HellfireOS*.

A seção 2 irá apresentar a estratégia adotada na versão paralela do algoritmo de resolução de labirintos, enquanto a seção 3 irá detalhar os resultados obtidos.

O código fonte pode ser encontrado na pasta hellfireos-master/app/trabalho_2.

2. Algoritmo Paralelo

Para paralelizarmos o algoritmo utilizamos o padrão de programação paralela mestre/escravo. O processador de identificador zero é o mestre, e os demais processadores são os escravos. No algoritmo, o mestre distribui um labirinto para cada um dos escravo, espera os escravos retornarem com os resultados e após isso, envia novamente um labirinto para cada um dos escravos. Isso é feito até que não se tenha mais labirintos para computar.

```
enquanto(existe_labirinto_para_ser_computador)
{
    para todo escravo
    {
        pega próximo labirinto
        envia labirinto para o escravo
    }
    para todo escravo
    {
        recebe resultado computado pelo escravo
    }
}
```

Por sua vez, cada escravo aguarda receber um labirinto do mestre, então o computa, e

após envia o resultado para o mestre.

```
enquanto(verdadeiro)
{
    recebe labirinto do mestre
    computa solução
    envia solução para o mestre
}
```

Na prática, o que temos é uma barreira de sincronização na função do mestre, cujo espera todos os escravos terminarem suas tarefas, para então distribuir novos labirintos, o que consiste em um escalonamento estático. O escalonamento dinâmico seria mais adequado para obter melhor performance, uma vez que a carga de computação de tarefa não é uniforme, no entanto, a exiguidade do momento não nos permite prosseguir com implementações mais robustas.

Algo importante a se notar, é que dividimos o envio das tarefas em duas partes. Primeiro o mestre manda a estrutura *maze_s*, espera um retorno de sinal do escravo, e após envia o labirinto. Tomamos essa escolha pelo motivo de que na estrutura *maze_s* temos apenas um ponteiro para o labirinto, de forma que ao enviar a estrutura através da rede, mandamos dados importantes como dimensões do labirinto e ponto inicial e ponto final, porém não mandamos o labirinto em si. Para resolver isso, mandamos a estrutura, esperamos um sinal de resposta do escravo, e após mandamos o labirinto. Esperar um sinal do escravo sinalizando que ele recebeu a estrutura é importante, pois os pacotes são datagramas, ou seja não garantem a entrega, o que implica no fato de que se o mestre enviar a estrutura e imediatamente o labirinto, pode acontecer dos dados dados serem perdidos.

```
envia estrutura maze_s para o escravo
aguarda receber sinal do escravo
envia labirinto para o escravo
```

Consequentemente, a atividade de recebimento do escravo também é dividida em duas partes.

```
recebe estrutura maze_s do mestre
envia sinal para o mestre
recebe labirinto do mestre
```

3. Avaliação de Desempenho

Realizamos experimentos utilizando a plataforma noc 3x2. Rodamos o algoritmo 10 vezes na sua versão sequencial, 10 vezes na sua versão paralela e coletamos a média aritmética simples.

A nossa versão paralela apresentou 4,8 de speedup executando o algoritmo com 6 processadores, ou seja, o speedup está próximo do ideal. No entanto, como o escalonamento não é dinâmico, acreditamos que a aceleração tende a diminuir ao aumentar o número de núcleos, e isso se deve ao tempo de espera na barreira, que deixam ociosos o mestre e os escravos que já concluíram suas tarefas.

4. Conclusão

Nossa implementação apresentou bom resultado para uma quantidade pequena de processadores, embora a implementação seja simples. Isso mostra na verdade, que paralelizar a computação de labirintos pode alcançar boa escalabilidade sem esforços brutais, uma vez que cada labirinto pode ser resolvido de maneira complementamente independente.

Questionamento que permanecem a despeito da conclusão do trabalho, são dois. Como enviar de uma só vez a estrutura *maze_s* e o labirinto propriamente dito, e como implementar uma estratégia de escalonamento eficiente.